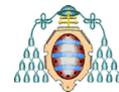


Pruebas del Software Parte 2

Sistemas de Información



Javier Tuya (tuya@uniovi.es)
Claudio de la Riva (claudio@uniovi.es)
Grupo de Investigación en Ingeniería del Software - <http://giis.uniovi.es>
Curso 2022-2023



1

Contenidos

- Introducción y definiciones
- Proceso de pruebas de software
- Técnicas básicas de prueba
 - Pruebas realizadas en asignatura anterior
 - Basadas en clases de equivalencia – Ejemplo
 - Modelo en V. Niveles y Tipos de prueba
- Ejemplo de prácticas de aula (Parte 2)
 - Automatización con Junit
 - Diseño e implementación
 - Obtención lista de carreras
 - Obtención descuentos/recargos
 - Otros...

2

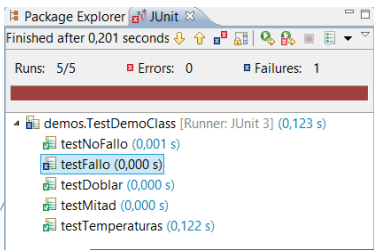
Automatización con JUnit

```
package demos;

import junit.framework.TestCase;

/* Diversas pruebas escritas en JUnit para mostrar su uso */
public class TestDemoClass extends TestCase {
    /** assert que nunca produce un fallo */
    public void testNoFallo() {
        assertEquals(1,1);
    }
    /** assert que siempre produce un fallo */
    public void testFallo() {
        assertEquals("fallo",1,2);
    }
    /** funcionamiento de doblar */
    public void testDoblar() {
        DemoClass c=new DemoClass();
        assertEquals(2,c.doblar(1));
        assertEquals(0,c.doblar(0));
    }
    /** funcionamiento de mitad y control de excepciones */
    public void testMitad() {
        DemoClass c=new DemoClass();
        assertEquals(1.0,c.dividir(2,2),0.0001);
        assertEquals(2.0,c.dividir(4,2),0.0001);
    }
}
```

- Framework para ejecución tests en Java, independiente e integrado en Eclipse
- Automatización prueba en desarrollo y regresión
- Clases derivadas de TestCase (autodocumentadas)
- Antes y después de cada método
 - setUp, tearDown
- Para .NET: NUnit, MS Test



Qué versión usamos?

3

Automatización con Junit (Comparación versiones)

JUnit 3

```
import junit.framework.*;
extend TestCase
```

Convenios:

- TestClass
- setUp()
- tearDown()
- testX()
- assertX(msg,exp,act)
- assertX(exp,act)

JUnit 4

```
import org.junit.*;
import static org.junit.Assert.*
```

Anotaciones:

- @Before
- @After
- @Test
- @BeforeClass
- @AfterClass
- assertX(msg,exp,act)
- assertX(exp,act)
- +test parametrizados (mejor JUnitParams), manejo excepciones...

JUnit 5

```
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*
```

Anotaciones:

- @BeforeEach
- @AfterEach
- @Test
- @BeforeAll
- @AfterAll
- assertX(exp,act,msg)
- assertX(exp,act)
- +test parametrizados similar JUnitParams, tests anidados, extensiones, lambdas, compatibilidad hacia atrás (JUnit Vintage)...

Hay otros frameworks para Java, p.e. TestNG

Para .Net: MSTest, Nunit, xUnit

4

Ejemplo

- <https://github.com/javiertuya/samples-test-dev>
- Javadoc: <https://javiertuya.github.io/samples-test-dev>

- Historias de usuario a probar relacionadas con:

- ☐ Inscripción de un atleta
- ☐ Ver javadoc de giis.demo.tkrun

(1) Como usuario quiero visualizar las carreras en las que está abierta la inscripción

(2) Como usuario quiero visualizar los datos detallados de la carrera seleccionada

samples-test-dev

- > src/main/java
- > src/test/java
- > giis.demo.jdbc.ut
- > giis.demo.tkrun.ut
 - > TestInscripcion.java
 - > TestInscripcionUnit3.java
 - > TestInscripcionUnitParams.java
 - > TestInscripcionParametrized.java
 - > TestUpdates.java
- > package.html
- > src/it/java
- > src/main/resources
- > src/test/resources
- > JRE System Library [JavaSE-1.8]
- > Maven Dependencies

Carreras

Simulación de la

Fecha de hoy (for

Proximas carreras:

id	descr	estado
101	en fase 3	(Abierta)
102	en fase 2	(Abierta)
103	en fase 1	(Abierta)
104	antes inscripcion	

La misma informacion que en la tabla, pero en forma de lista/combo

101-en fase 3 (Abierta)

Al seleccionar la tabla (no el combo) muestra detalles

Porcentaje de descuento: 0%

id	102
inicio	2016-11-05
fin	2016-11-09
fecha	2016-11-20
descr	en fase 2

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

5

5

Diseño e implementación

(1) Obtención lista carreras

- Vamos a realizar **pruebas unitarias**

- ☐ Probar las funciones de los procesos de negocio (a nivel de modelo en MVC)

- (1) Como usuario quiero visualizar las carreras en las que está abierta la inscripción.

- ☐ Podré inscribirme a una carrera entre las fechas de inicio y fin establecidas y también posteriormente hasta el día de la carrera (inclusive). Visualizaré el id y descripción de todas las carreras excluyendo las pasadas, con la indicación del estado Abierto en las que se puede realizar inscripción
- ☐ La función que muestra la lista de selección
 - List **getListaCarreras**(Date fechaInscripcion)

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

6

6



(1) Obtención lista carreras Clases de Equivalencia

- Lista de carreras (**getListaCarreras**)
 - Fecha de inscripción (entradas)
 - Periodos válidos
 - Antes de la fecha tope (Plazo I)
 - Después de la fecha tope y antes de la fecha de la prueba (Plazo II)
 - El día de la prueba (Plazo III)
 - Periodos inválidos
 - Antes de la apertura de inscripción
 - Después de la prueba
 - Carreras que se muestran en la lista (clases de equivalencia para las salidas)
 - Los mismos periodos que para Fecha inscripción

En esta función todos los periodos son válidos (en este caso el comportamiento es que estas filas no se mostrarán en la salida)

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

7

7



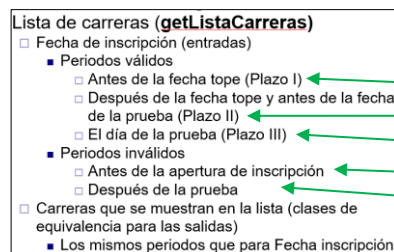
(1) Obtención lista carreras Casos de prueba

- Entradas del caso de prueba:
 - Fecha de inscripción
 - La Base de Datos
- Salidas del caso de prueba:
 - El listado de carreras

■ Cuántos casos de prueba?

- Cubrir las 5 clases anteriores:

- 5 casos de prueba, con 5 fechas diferentes y una única fila en la BD



Caso
1
2
3
4
5

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

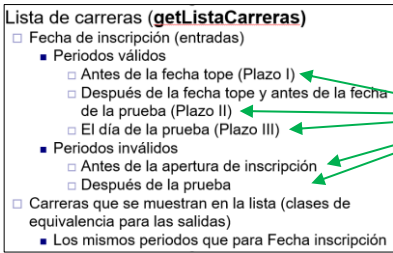
8

8

(1) Obtención lista carreras

Casos de prueba

- Pero la Base de Datos también es ENTRADA
- Cubrir las 5 clases:
 - 1 caso de prueba con 1 fecha de inscripción y 5 filas en la BD
- Automatizaremos con Junit 4
- Definiremos la **inicialización** de estos datos
- La **ejecución** del caso de prueba
- Y la **comparación** de resultados



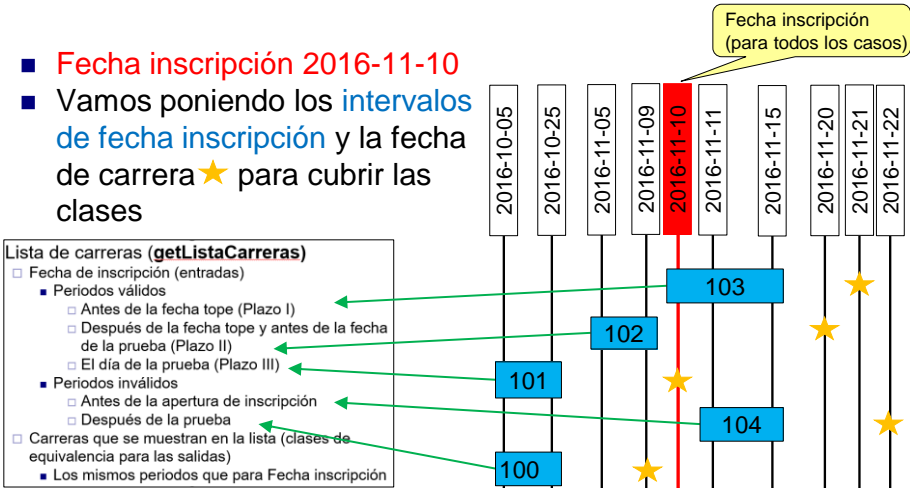
Caso

TestInscripcion.
testCarrerasActivas*

(1) Obtención lista carreras

Inicialización de datos

- Fecha inscripción 2016-11-10
- Vamos poniendo los intervalos de fecha inscripción y la fecha de carrera ★ para cubrir las clases



(1) Obtención lista carreras

Inicialización de datos

- El método **setUp** se ejecuta antes de CADA UNO de los tests
- Creamos los datos de la BD necesarios para las situaciones a cubrir por nuestro test (5 rangos de fechas)
- Hemos aprovechado a poner el valor de cada clase en los **límites de cada rango**

```
public class TestInscripcion {
    private static Database db=new Database();
    @Before
    public void setUp() {
        db.createDatabase(true);
        loadCleanDatabase(db);
    }
    @After
    public void tearDown(){}
    public static void loadCleanDatabase(Database db) {
        db.executeBatch(new String[] {
            "delete from carreras",
            "insert into carreras(id,inicio,fin,fecha,descr) values (100,'2016-10-05','2016-10-25','2016-11-09','finalizada')",
            "insert into carreras(id,inicio,fin,fecha,descr) values (101,'2016-10-05','2016-10-25','2016-11-10','en fase 3')",
            "insert into carreras(id,inicio,fin,fecha,descr) values (102,'2016-11-05','2016-11-09','2016-11-20','en fase 2')",
            "insert into carreras(id,inicio,fin,fecha,descr) values (103,'2016-11-10','2016-11-15','2016-11-21','en fase 1')",
            "insert into carreras(id,inicio,fin,fecha,descr) values (104,'2016-11-11','2016-11-15','2016-11-22','antes inscripcion')";
        });
    }
}
```

Objeto para uso en todos los test

@Before indica que se el método ejecutará justo antes de cada test, Asegurando que la inicialización de la base de datos es predecible y que los casos son independientes

Datos diseñados específicamente para cubrir cada situación, Dada una fecha de inscripción

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

11

11

(1) Obtención lista carreras

Comparación de resultados

- Asserts sobre los ítems que componen la lista de carreras

```
@Test
public void testCarrerasActivasList() {
    CarrerasModel inscr=new CarrerasModel();
    List<Object[]> carreras=inscr.getListCarrerasArray(Util.isoStringToDate("2016-11-10"));
    //Deben mostrarse todas las carreras de la BD menos la primera que es pasada, la ultima n
    assertEquals("el numero de carreras mostradas es incorrecto",4,carreras.size());
    //la lista de carreras contiene un array de objetos de una dimension
    assertEquals("101-en fase 3 (Abierta)",carreras.get(0)[0]);
    assertEquals("102-en fase 2 (Abierta)",carreras.get(1)[0]);
    assertEquals("103-en fase 1 (Abierta)",carreras.get(2)[0]);
    assertEquals("104-antes inscripcion ",carreras.get(3)[0]);
}
```

La etiqueta que indica que es un caso de prueba

Los métodos assert* son estáticos, derivan de org.junit.Assert. Especificar el nombre de la clase Assert para descubrir otras posibilidades (assertTrue, assertNull,)

Assert indicando un texto adicional (permite autodocumentar y facilita diagnostico)

Se compara cada ítem del array. Si el método de prueba fuese el que obtiene una lista de DAOs, se debería comprobar cada atributo de cada ítem

Visualización de un fallo y el stacktrace. El texto del assert facilita la identificación del fallo

Package Explorer

Finished after 1,244 seconds

Runs: 1/1 Errors: 0 Failures: 1

testCarrerasActivasList [Runner: JUnit 4] (1,186 s)

Failure Trace

java.lang.AssertionError: el numero de carreras mostradas es incorrecto expected:<4> but was:<3>

at gilis.demo.tkrunit.TestInscripcion.testCarrerasActivasList(TestInscripcion.java:65)

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

12

12

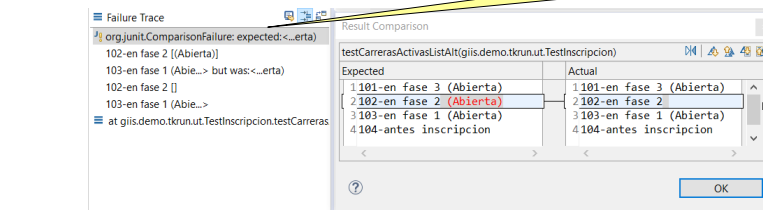
(1) Obtención lista carreras

Alternativa para comparación

- Cuando comparamos cada elemento de una lista, si falla uno, no sabemos qué pasa con los siguientes
- Ahora comparamos todos los elementos de la lista de una vez usando un método que pasa la lista a un string (**facilita la comparación**)

```
62 @Test
63 public void testCarrerasActivasListAlt() {
64     CarrerasModel inscr=new CarrerasModel();
65     List<Object[]> carreras=inscr.getListaCarrerasArray(Util.isoStringToDate("2016-11-10"));
66     assertEquals(
67         "101-en fase 3 (Abierta)\n102-en fase 2 (Abierta)\n103-en fase 1 (Abierta)\n104-antes inscripcion ",
68         list2string(carreras));
69 }
```

Doble click muestra detalles de las diferencias



J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

13

13

(1) Obtención lista carreras

Validacion y excepciones

- Todas las clases diseñadas son válidas, pero el método recibe la fecha actual como parámetro.
- Añadimos test para validación de que no sea un objeto nulo.
- En este caso **el comportamiento esperado es que produzca excepción**

Forma más simple, indico qué tipo de excepción debe producirse al final del método para que pase el test

```
138 @Test(expected=ApplicationException.class)
139 public void testCarrerasActivasException1() {
140     CarrerasModel inscr=new CarrerasModel();
141     inscr.getListaCarreras(null);
142 }
143 * Utilizacion de reglas, para comprobar tambien los mensajes de las ex
144 @Rule
145 public ExpectedException thrown = ExpectedException.none();
146 @Test
147 public void testCarrerasActivasException2() {
148     CarrerasModel inscr=new CarrerasModel();
149     thrown.expect(ApplicationException.class);
150     thrown.expectMessage("La fecha de inscripcion no puede ser nula");
151     inscr.getListaCarreras(null);
152 }
153
154 }
```

Utilizando esta regla, permite comprobar diferentes tipos de excepciones y los mensajes

MUY IMPORTANTE

- Asserts sobre excepciones solo cuando el comportamiento esperado es excepción.
- Nunca filtrar excepciones del código que estamos probando.

Ver y comparar variantes 1 y 3 en el código

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

14

14

Diseño e implementación

(2) Porcentaje de descuento

- (2) Como usuario quiero **visualizar los datos detallados de la carrera seleccionada**.
 - Si realizo la inscripción en las fechas de inscripción establecidas se aplica un descuento del 30%, si es después se aplica 0% y el día de la carrera un recargo del 50%. Cuando seleccione una carrera visualizaré todos los detalles de ésta y el porcentaje de descuento o recargo aplicable en función de la fecha de hoy. Cuando cambie la fecha de hoy se actualizará la tabla con las carreras activas, manteniendo la selección previa los detalles correspondientes.
 - La función para determinar el descuento/recargo de la inscripción a una prueba
 - `int getDescuentoRecargo(long idCarrera, Date fechaInscripcion)`

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

15

15

(2) Porcentaje de descuento

Clases de equivalencia

- Porcentaje de descuento (`getDescuentoRecargo`)
 - Fecha de inscripción (entradas)
 - Periodos válidos
 - Antes de la fecha tope (Plazo I)
 - Después de la fecha tope y antes de la fecha de la prueba (Plazo II)
 - El día de la prueba (Plazo III)
 - Periodos inválidos
 - Antes de la apertura de inscripción (**inválida**)
 - Después de la prueba (**inválida**)
 - Carrera para la que se solicita el descuento/recargo
 - Id existente
 - Id no existente (**inválida**)

Las clases de equivalencia son muy similares al anterior, pero ahora hay clases inválidas

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

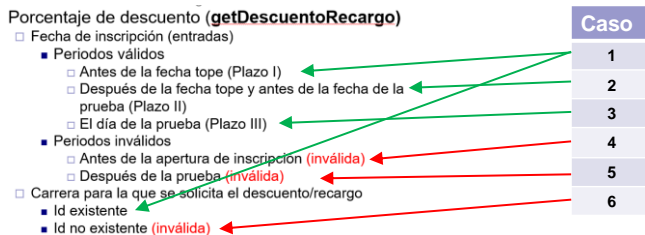
16

16

(2) Porcentaje de descuento

Casos de prueba

- Entradas del caso de prueba:
 - Fecha de inscripción
 - La Base de Datos
- Salidas del caso de prueba:
 - Porcentaje descuento/recargo
- **Cuántos casos de prueba?**
- Cubrir las clases anteriores:
 - Necesitaremos un caso de prueba por cada una
 - Pero **reutilizamos** la BD establecida en el anterior ejemplo



J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

17

17

(2) Porcentaje de descuento

Varios casos de prueba en un método

- Prueba del porcentaje de descuento (clases válidas):
 - Cubre **todas las clases válidas** (3 relativas fecha de inscripción y 1 relativa al id de carrera)
 - Añadiremos 2 más para probar los límites de los valores de los rangos
 - Notar que en realidad estamos ejecutando cinco casos de prueba en un único método
 - Si falla uno de ellos, el resto de casos no se ejecutan
 - Si ponemos 5 métodos de prueba, podemos tener demasiado código

```

161 @Test
162 public void testPorcentajeDescuentoRecargoValidas() {
163     //Reutilizamos el setUp para los tests de la lista de carreras mostradas al usuario
164     //utilizando una fecha y diferentes carreras que nos cubran las clases validas
165     Date fecha=Util.isoStringToDate("2016-11-10");
166     CarrerasModel inscr=new CarrerasModel();
167     assertEquals(-30,inscr.getDescuentoRecargo(103,fecha));
168     assertEquals(0,inscr.getDescuentoRecargo(102,fecha));
169     assertEquals(+50,inscr.getDescuentoRecargo(101,fecha));
170     //Como no se han probado los valores limite en los dos extremos de los rangos,
171     //anyade casos para ello (fase 1 y 2, extremo superior)
172     assertEquals(-30,inscr.getDescuentoRecargo(103,Util.isoStringToDate("2016-11-15")));
173     assertEquals(0,inscr.getDescuentoRecargo(102,Util.isoStringToDate("2016-11-19")));
174 }

```

Con estas pruebas se comprueban los limites superiores de los rangos

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

18

18

(2) Porcentaje de descuento

Clases inválidas - excepciones

- Prueba del porcentaje de descuento (clases inválidas):
 - El **comportamiento esperado es una excepción**
 - Son tres métodos muy similares, creamos un método genérico con parámetros que invocamos desde cada uno de los tests
 - (algo similar podríamos haber hecho en el anterior, pero aquí es imprescindible porque las excepciones se tienen que probar por separado)
 - @Rule ya ha sido definida en la clase con anterioridad

```
183 @Test public void testPorcentajeDescuentoRecargoInvalidaCarreraFinalizada() {
184     porcentajeDescuentoRecargoInvalidas(100, "No es posible la inscripcion en esta fecha");
185 }
186 @Test public void testPorcentajeDescuentoRecargoInvalidaCarreraAntesInscripcion() {
187     porcentajeDescuentoRecargoInvalidas(104, "No es posible la inscripcion en esta fecha");
188 }
189 @Test public void testPorcentajeDescuentoRecargoInvalidaCarreraNoExiste() {
190     porcentajeDescuentoRecargoInvalidas(99, "Id de carrera no encontrado: 99");
191 }
192 public void porcentajeDescuentoRecargoInvalidas(long idCarrera, String message) {
193     Date fecha=Util.isoStringToDate("2016-11-10");
194     CarrerasModel inscr=new CarrerasModel();
195     thrown.expect(RuntimeException.class);
196     thrown.expectMessage(message);
197     inscr.getDescuentoRecargo(idCarrera, fecha);
198 }
---
```

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

19

19

(2) Porcentaje de descuento

Pruebas parametrizadas

- Un único método de prueba se ejecutará varias veces con diferentes parámetros (muy útil si son muchos o el método de prueba complejo)

Indicar un runner específico para pruebas parametrizadas

Valores de los parámetros para cada uno de los tests a ejecutar

Mapeo de la posición de cada parámetro a variables en el método de prueba

```
@RunWith(Parameterized.class)
public class TestInscripcionParametrized {
    private static Database db=new Database();
    public void setUp() {}
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{
            {"2016-11-10", -30, 103},
            {"2016-11-10", 0, 102},
            {"2016-11-10", +50, 101},
            {"2016-11-15", -30, 103},
            {"2016-11-19", 0, 102},
        });
    }
    @Parameter(value=0) public String fechaStr;
    @Parameter(value=1) public int descuentoRecargo;
    @Parameter(value=2) public long idCarrera;
    @Test
    public void testPorcentajeDescuentoRecargoValidas() {
        Date fecha=Util.isoStringToDate(fechaStr);
        CarrerasModel inscr=new CarrerasModel();
        assertEquals(descuentoRecargo, inscr.getDescuentoRecargo(idCarrera, fecha));
    }
}
```

Runs: 5/5 Errors: 0 Failures: 0

giis.demo.tkrun.ut.TestInscripcionParametrized [Runner: JU]

[0] (1,844 s)

testPorcentajeDescuentoRecargoValidas[0] (1,844 s)

[1] (0,188 s)

[2] (0,222 s)

[3] (0,272 s)

[4] (0,247 s)

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

20

20

(2) Porcentaje de descuento

Pruebas parametrizadas (mejor)

- Las pruebas parametrizadas de JUnit 4 restringen a un único conjunto de parámetros en la clase, y requieren código adicional (JUnit 5 lo mejora).
- El componente JUnitParams simplifica todo (añadir dependencia Maven)
- Restricción: ejecutar todos los tests de la clase

Runner específico de JUnitParams

```
@RunWith(JUnitParamsRunner.class)
public class TestInscripcionJUnitParams {
    private static Database db=new Database();
    public void setUp() {}
    @Test
    @Parameters({
        "2016-11-10, -30, 103",
        "2016-11-10, 0, 102",
        "2016-11-10, +50, 101",
        "2016-11-15, -30, 103",
        "2016-11-19, 0, 102"})
    public void testPorcentajeDescuentoRecargoValidas(String fechaStr, int descuentoRecargo, long idCarrera) {
        Date fecha=Util.isoStringToDate(fechaStr);
        CarrerasModel inscr=new CarrerasModel();
        assertEquals(descuentoRecargo,inscr.getDescuentoRecargo(idCarrera,fecha));
    }
}
```

Valores de los parámetros array formato csv (más simple)

El método de test recibe los parámetros directamente

Runs: 5/5 Errors: 0 Failures: 0

glis.demo.tkrun.ut.TestInscripcionJUnitParams [Runner: JUnit 4] (2,247 s)

- testPorcentajeDescuentoRecargoValidas (2,247 s)
- testPorcentajeDescuentoRecargoValidas(2016-11-10, -30, 103) [0] (1,289 s)
- testPorcentajeDescuentoRecargoValidas(2016-11-10, 0, 102) [1] (0,195 s)
- testPorcentajeDescuentoRecargoValidas(2016-11-10, +50, 101) [2] (0,242 s)
- testPorcentajeDescuentoRecargoValidas(2016-11-15, -30, 103) [3] (0,257 s)
- testPorcentajeDescuentoRecargoValidas(2016-11-19, 0, 102) [4] (0,264 s)

J Tuya, 2022-2023 SI - Pruebas del Software - Parte 2 21

21

Comprobaciones cuando se actualiza la Base de Datos

- Nuevo método (actualiza inicio/fin de inscripción)
 - **updateFechasInscripcion**(int idCarrera, Date inicio, Date fin)
- El nuevo estado de la BD es también salida. Principio fundamental:
 - Comprobar que **hace lo que debe hacer**
 - Y que **no hace lo que no debe hacer**
- Ejemplo test que actualiza fechas de carrera 101:

```
27@
28 @Test
29 public void testUpdateFechasInscripcion1() {
30     CarrerasModel inscr=new CarrerasModel();
31     inscr.updateFechasInscripcion(101, Util.isoStringToDate("2016-09-01"), Util.isoStringToDate("2016-09-02"));
32     //el test habra modificado las dos fechas de la carrera 101,
33     //lee todos los datos de la tabla y las compara con los iniciales tras cambiar solamente estos dos datos
34     List<CarreraEntity> carreras=db.executeQueryPojo(CarreraEntity.class, "SELECT * FROM carreras ORDER BY id");
35     assertEquals(
36         "100,2016-10-05,2016-10-25,2016-11-09,finalizada\n"
37         +"101,2016-09-01,2016-09-02,2016-11-10,en fase 3\n"
38         +"102,2016-11-05,2016-11-09,2016-11-20,en fase 2\n"
39         +"103,2016-11-10,2016-11-15,2016-11-21,en fase 1\n"
40         +"104,2016-11-11,2016-11-15,2016-11-22,antes inscripcion\n",
41         Util.pojosToCsv(carreras,new String[] {"id","inicio","fin","fecha","descr"}));
}
```

Leemos el estado final de la BD y comprobamos los valores de la tabla

J Tuya, 2022-2023 SI - Pruebas del Software - Parte 2 22

22

Comprobaciones cuando se actualiza la Base de Datos

- Problema de la anterior implementación:
 - Hay que poner demasiados datos en la salida deseada
- Otra forma más práctica:
 - Leer los datos que no cambiarán de la base de datos
 - Especificar solo los que cambiarán

```
@Test
public void testUpdateFechasInscripcion2() {
    List<CarreraEntity> expected=db.executeQueryPojo(CarreraEntity.class, "SELECT * FROM carreras ORDER BY id");
    expected.get(1).setInicio("2016-09-01");
    expected.get(1).setFin("2016-09-02");

    CarrerasModel inscr=new CarrerasModel();
    inscr.updateFechasInscripcion(101, Util.isoStringToDate("2016-09-01"), Util.isoStringToDate("2016-09-02"));

    List<CarreraEntity> actual=db.executeQueryPojo(CarreraEntity.class, "SELECT * FROM carreras ORDER BY id");
    assertEquals(Util.pojosToCsv(expected,new String[] {"id","inicio","fin","fecha","descr"}),
        Util.pojosToCsv(actual,new String[] {"id","inicio","fin","fecha","descr"}));
}
```

Antes crea la salida deseada, incluyendo los cambios que deberán realizar el método que estamos probando

Faltaría diseñar pruebas para validar los intervalos de fechas y parametros

La comparación con el assert se simplifica

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

23

23

Diseño e implementación (3) Otras Historias de Usuario

- Prueba de la **realización de la inscripción**:
 - En este caso la salida son los datos creados en la BD
 - Diseñamos e implementamos nuevos tests:
 - Reutilizando los datos del setUp de la base de datos
 - Añadiendo los datos adicionales que necesitemos
 - Para comparar los resultados
 - Obtener el último id de inscripción antes de probar
 - Tras la inscripción comprobar el estado final de la base de datos: que se ha creado la inscripción (como en anterior)
 - y que la nueva inscripción tiene el id posterior al último id
- Comprobación de **inscripciones duplicadas**:
 - Reutilizamos lo anterior (clase de equivalencia: inscripción válida)
 - Para la nueva clase de equivalencia: inscripción duplicada
 - Ejecutar de nuevo una inscripción ya realizada
 - Comprobar la detección del duplicado (p.e. una excepción)
 - Comprobar que en la base de datos no ha cambiado nada

Discutir setup base datos:
@Before, @After
@BeforeClass @AfterClass
En cada test

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

24

24

Diseño e implementación

(3) Otras Historias de Usuario

- En otra historia de usuario posterior debemos añadir las funcionalidades de comprobación de modalidad y edad:

- ☐ Edad del solicitante
 - Mayor de edad
 - Menor de edad (inválida)
- ☐ Modalidad de inscripción
 - Elite
 - Popular
 - Cualquier otra diferente a las anteriores (inválida)
- ☐ Nacionalidad del solicitante
 - Española
 - Otra

- Automatizamos las pruebas:

- ☐ Completando los datos del setup
- ☐ Reutilizando pruebas existentes
- ☐ Completando pruebas existentes
- ☐ Creando nuevos métodos de prueba

Para cada historia de usuario podemos definir las pruebas ANTES DE TENER ESCRITO EL CODIGO DE LA APLICACIÓN

Luego se va incorporando el código para pasen los tests

Esto se denomina TDD (Test Driven Development)

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

25

25

Resumen

- Actividad más utilizada para el aseguramiento de calidad del software
 - ☐ Hems visto su uso par evaluación del software para emitir un veredicto
 - Encontrar fallos para poder solucionar los defectos y aumentar la confianza
 - No es lo mismo que depuración
 - ☐ Técnicas y criterios de prueba: Diseñar **buenos casos de prueba**
 - ☐ Para el diseño de pruebas: Utilización de todos los artefactos software disponibles (test basis)
 - ☐ Automatización con JUnit: **Formar parte de la rutina diaria, al igual que Git**
- Más (En el ejemplo hemos diseñado y automatizado pruebas unitarias, usando la técnica de clases de equivalencia y un enfoque black-box)
 - ☐ Probar las validaciones de datos en los formularios
 - ☐ Prueba integrada desde el UI (p.e. con Selenium)
 - ☐ Otros tipos de pruebas (p.e. volúmenes grandes de datos, seguridad...)
 - ☐ No todas las pruebas pueden ser automatizadas
 - ☐ Otros aspectos (técnicas, gestión...)
- Posteriores asignaturas:
 - ☐ Cuarto: Pruebas y Despliegue de Software
 - ☐ Máster: Calidad de Procesos y Productos

J Tuya, 2022-2023

SI - Pruebas del Software - Parte 2

26

26